Comment *bien* écrire une requête SQL

Rappel : la syntaxe `--` introduit un *commentaire* en SQL (ce qui suit est ignoré).

1. Structure générale d'une requête

1.1. Respecter strictement l'ordre des clauses

- Pour une requête de sélection : `SELECT` → `FROM` → `JOIN` (si besoin) → `WHERE` → `ORDER BY` (→ `LIMIT` si nécessaire).

```sql
SELECT nom_colonne1, nom_colonne2
FROM table
JOIN autre_table ON condition_jointure -- seulement si jointure nécessaire !
WHERE condition_filtre
ORDER BY nom_colonne1
LIMIT 10 ; -- pour limiter la quantité de lignes renvoyées
```

- Pour une requête de mise à jour (voir plus bas) : `UPDATE` → `SET` → `WHERE`.

```sql
UPDATE Table
SET colonne1 = nouvelle_valeur1, colonne2 = nouvelle_valeur2
WHERE condition ;
```

> ⚠️ **Attention : l'ordre est imposé !**
>
> `WHERE` vient toujours après `FROM` / `JOIN`, `ORDER BY` en dernier.
> De même, le `SET` vient avant le `WHERE`.

1.2. Quand utiliser JOIN ?

RÈGLE

- Si toutes les colonnes sélectionnées **et** les conditions de filtrage (`WHERE`) viennent d'**une seule et *même* table**, `JOIN` est *inutile*.

  Exemple :

  ```
  SELECT titre FROM Album
  WHERE annee > 2000 ;
  -- titre et annee sont des champs de la table Album, pas besoin de JOIN
  ```

- Si les données nécessaires viennent de **tables différentes *reliées***, une jointure est **nécessaire.** Il faut alors préfixer les noms de colonnes ambigus par le nom de la table (voir plus bas la section « Qualification des colonnes ambiguës »).

  Exemple :

  ```
  SELECT Chanson.titre FROM Chanson
  JOIN Album ON Chanson.idalbum = Album.id
  WHERE Album.annee > 2000 ;
  -- le filtre porte sur la table Album, pas Chanson !
  ```

2. Utilisation des guillemets simples pour les chaînes de caractères

- **OBLIGATOIRE** pour les valeurs textuelles : `WHERE nom = 'Muse'`.
- **INTERDIT** pour les nombres : `WHERE id = 10` (*pas* `'10'`).
- **INTERDIT** pour les noms de colonnes/tables : `SELECT titre` (*pas* `'titre'`).

3. Syntaxe des jointures (JOIN moderne)

✅ **À FAIRE** : syntaxe explicite avec `JOIN ... ON`.

```
FROM Chanson
JOIN Album ON Chanson.idalbum = Album.id
```

⛔ **À ÉVITER** : syntaxe *obsolète* avec des virgules.

```
FROM Chanson, Album
WHERE Chanson.idalbum = Album.id
```

Raison : la syntaxe moderne sépare clairement les *jointures* (`ON`) des *filtres* (`WHERE`).

4. Qualification des colonnes ambiguës

Lorsque plusieurs tables ont des colonnes de même nom, **toujours qualifier par le nom de la table** !

Ainsi, en supposant que les tables `Chanson` et `Album` ont une colonne portant le même nom `titre`, la requête suivante est ambiguë, et peut provoquer une erreur.

⛔ **ERREUR** :

```
SELECT titre FROM Chanson JOIN Album ON ... ;
-- Quel `titre` nous intéresse : `Chanson.titre` ou `Album.titre` ?
```

✅ **CORRECT** :

```
SELECT Chanson.titre, Album.titre FROM Chanson JOIN Album ON ... ;
```

5. Conditions de jointure complètes

Vérifier que chaque `JOIN` a sa condition `ON` :

```
FROM Chanson
JOIN Album ON Chanson.idalbum = Album.id
JOIN Groupe ON Album.idgroupe = Groupe.id  -- Ne pas oublier !
```

> ⚠️ **Oublier un `ON` ⇒ risque de produit cartésien !**
>
> En cas d'oubli de `ON`, *toutes les combinaisons possibles* seront formées. Cela amènera une *explosion* du nombre de résultats, d'où une efficacité réduite des requêtes. Le serveur pourra s'en trouver ralenti et sa consommation électrique augmentera.

6. Fin de requête avec point-virgule

Toujours terminer par un point-virgule :

```
SELECT * FROM Chanson WHERE titre = 'Showbiz' ;
```

C'est une bonne pratique, même si certains SGBD peuvent faire « autrement ».

7. Mots-clés SQL en MAJUSCULES

Convention de lisibilité (pas obligatoire techniquement, mais recommandée) :

```
SELECT titre FROM Album WHERE annee > 2000 ;
```

8. Clause WHERE : ET logique et OU logique

- `AND` : toutes les conditions doivent être vraies.
- `OR` : au moins une condition doit être vraie.
- Utiliser des parenthèses si nécessaire : `WHERE (a = 1 OR a = 2) AND b = 3`.

9. ORDER BY pour trier les résultats

9.1. Tri par défaut : ordre croissant (ASC)

- Par défaut, `ORDER BY` trie en **ordre croissant** :

```
SELECT * FROM Album ORDER BY titre ;
-- Tri alphabétique : A, B, C...

SELECT * FROM Album ORDER BY annee ;
-- Tri numérique : 1999, 2000, 2001...
```

- Le mot-clé `ASC` (ascendant) est *optionnel* car c'est le comportement par défaut :

```
ORDER BY titre ; -- Équivalent à ORDER BY titre ASC ;
```

- Un nom de champ **doit** suivre la clause `ORDER BY`.

9.2. Modifier le tri : ordre décroissant (DESC)

Pour trier en **ordre décroissant**, ajouter `DESC` après le nom de la colonne :

```
SELECT * FROM Album ORDER BY annee DESC ;
-- Du plus récent au plus ancien : 2003, 2002, 2001...

SELECT * FROM Album ORDER BY titre DESC ;
-- Tri alphabétique inverse : Z, Y, X...
```

9.3. Tri sur plusieurs colonnes (encore jamais vu au bac)

On peut trier sur *plusieurs* critères successifs :

```
SELECT * FROM Chanson
ORDER BY album ASC, titre DESC ;
-- D'abord par album (croissant), puis par titre (décroissant) dans chaque album
```

9.4. Après ORDER BY : LIMIT pour limiter le nombre de résultats (hors programme au sens strict)

`LIMIT` permet de ne récupérer que les *N* premiers résultats d'une requête :

```
SELECT * FROM Chanson
ORDER BY annee DESC LIMIT 5 ; -- Récupère les 5 chansons les plus récentes
```

- C'est utile pour tester des requêtes (on éviterait ainsi de récupérer 10 000 lignes, si tel devait être le résultat réel !)
- La clause `LIMIT` vient *toujours en dernier* (après `ORDER BY`).

10. Concernant le mot-clé AS (alias)

10.1. Utilisation pour les tables

Remarque : vu au bac, cf. exercice 3 du sujet 24_NSIJ2JA1 (au moins).

```
FROM Album AS a
JOIN Groupe AS g ON a.idgroupe = g.id
```

- Le mot-clé `AS` est **optionnel** : `FROM Album a` fonctionne aussi.
- Les alias raccourcissent les requêtes et améliorent la lisibilité.
- Une fois l'alias défini, **utiliser uniquement l'alias** dans la suite de la requête.

10.2. Utilisation pour les colonnes

Remarque : vu au bac, cf. exercice 3 du sujet 24_NSIJ2JA1 (au moins).

```
SELECT COUNT(*) AS total, Album.titre AS nom_album
```

- Dans la réponse à la requête, le résultat du `COUNT(*)` se trouve nommé « `total` ».
- C'est utile pour donner des noms explicites aux « résultats des calculs » dont sont capables les SGBD (`COUNT`, `SUM`, `AVG`, etc. — on parle de *fonctions d'agrégation*).
- **Avantage** : facilite l'accès au résultat depuis un langage de programmation (PHP, Python, ...).
- `AS` est également **optionnel** pour les colonnes, mais recommandé pour la clarté.

> ⚠️ **Attention PIÈGE ☠️ !**
>
> Un alias défini avec `AS` **ne** peut **pas** être réutilisé dans la clause `WHERE` de la même requête (mais c'est possible dans la clause `ORDER BY`). Plus de détails dans la section suivante.

10.3. Alias, renommage et PIÈGE ☠️ !

Pour bien utiliser le mot-clef `AS`, mieux vaut bien connaître **l'ordre dans lequel sont exécutées** les diverses clauses d'une requête SQL.

1. Choix des tables (`FROM`, `JOIN`).
2. Choix des lignes (`ON`, `WHERE`).
3. Choix des colonnes (`SELECT`).
4. Traitement des colonnes (calcul ou fonction d'agrégation comme `AVG`).
5. Traitement des résultats (`DISTINCT`, `ORDER BY`).

Par conséquent, un renommage déclenché dans une clause `SELECT` **ne** peut **pas** être utilisé dans une clause `WHERE`. En effet, `WHERE` est en réalité traité *avant* `SELECT` !

Mais ce renommage peut être utilisé dans `ORDER BY`, qui est pris en compte *après* `SELECT`.

Exemples

- Requête **incorrecte** :

```sql
SELECT colonne1 AS c1 FROM table WHERE c1 > 10 ;
```

- Requête *correcte* :

```sql
SELECT colonne1 AS c1 FROM table ORDER BY c1 ;
```

> ⚠️ **Conduite à tenir vis-à-vis du renommage**
>
> **Dans le doute, si vous ne maîtrisez pas, N'utilisez PAS le renommage !!!**
>
> En revanche, comme des sujets de bac l'emploient, mieux vaut en connaître la signification. Rien ne vous oblige pour autant à l'utiliser dans vos réponses.

11. Requêtes de modification

11.1. Insertion de nouvelles données (INSERT INTO)

Syntaxe complète, **avec spécification** (de la valeur) **de la clef primaire** (approche maladroite !) :

```sql
INSERT INTO Chanson
VALUES (10, 'Megalomania', 'Hullabaloo', 'Muse') ;
```

C'est maladroit car *la clef primaire est souvent automatiquement incrémentée* par le SGBD. Voici une meilleure approche, en ne spécifiant **que** les champs à définir (certains, soumis à la **contrainte d'intégrité** `NOT NULL`, doivent être obligatoirement définis).

```
INSERT INTO Chanson (titre, album, groupe)
VALUES ('Megalomania', 'Hullabaloo', 'Muse') ;
```

⚠️ **Attention**

- Guillemets simples obligatoires pour les chaînes.
- Respecter l'ordre des colonnes.
- Si l'on ne veut pas gérer la clef primaire, il **faut** préciser les noms des champs qui seront impactés.
- L'ordre des champs dans `INSERT INTO` **n'**est **pas** libre : les valeurs dans `VALUES` doivent correspondre **exactement** à l'ordre des colonnes spécifiées.

11.2. Mise à jour d'enregistrements (UPDATE)

```
UPDATE Table
SET colonne1 = nouvelle_valeur1, colonne2 = nouvelle_valeur2
WHERE condition ;
```

⚠️ **Attention**

- Ordre à respecter : `UPDATE` → `SET` → `WHERE`.
- 💀 **DANGER** : oublier `WHERE` → modification de **TOUTE** la table *(catastrophe probable !)* **Exemple :** vous voulez changer le nom du groupe 'Muse' pour 'MUSE' (en majuscules). ⛔ Sans `WHERE` : `UPDATE Groupe SET nom = 'MUSE' ;` aura pour effet que **tous** les groupes s'appelleront ensuite 'MUSE' (plus de 'Radiohead', 'Coldplay', etc. → tous 'MUSE') ! ✅ Avec `WHERE` : `UPDATE Groupe SET nom = 'MUSE' WHERE nom = 'Muse' ;` → seul le groupe Muse est renommé.
- Guillemets simples pour les chaînes.
- Si plusieurs colonnes sont concernées, il faut les séparer par des virgules.

12. Considérations avancées

Les personnes curieuses ou désireuses de poursuivre des études supérieures en informatique *peuvent*

lire ce paragraphe (ça n'est pas une obligation pour autant).

12.1. Ordre des tables dans FROM avec JOIN (jointures internes)

Pour les jointures internes (`JOIN`), l'ordre des tables dans `FROM` n'a **pas d'importance** sur le résultat final.

Ces jointures sont **commutatives et associatives** : le moteur SQL les optimise et les réorganise automatiquement (il choisit lui-même l'ordre d'exécution le plus efficace — indépendamment de l'ordre dans lequel vous écrivez vos `JOIN`).

Commutativité :

```
FROM Chanson JOIN Album ON Chanson.idalbum = Album.id -- Ces deux requêtes
FROM Album JOIN Chanson ON Album.id = Chanson.idalbum -- sont équivalentes
```

Associativité :

```
-- Ces deux ordres de jointure produisent le même résultat :
FROM Chanson
JOIN Album ON Chanson.idalbum = Album.id
JOIN Groupe ON Album.idgroupe = Groupe.id

FROM Groupe
JOIN Album ON Album.idgroupe = Groupe.id
JOIN Chanson ON Chanson.idalbum = Album.id
```

Pour le bac, vous pouvez vous en tenir à cela. Si vous êtes curieux ou souhaitez poursuivre des études d'informatique, vous *pouvez* lire la suite !

12.2. ⚠️ Jointures externes (LEFT/RIGHT JOIN)

⚠️ Ces jointures sont *hors programme* ⚠️

Avec `LEFT JOIN`, l'ordre devient **crucial** car ces jointures ne sont **pas commutatives**. Elles conservent toutes les lignes de la table de **gauche** (celle avant le `LEFT JOIN`). Vous verrez cela dans le supérieur...

```
FROM Groupe LEFT JOIN Album ON ...  -- Garde TOUS les groupes et y « ajoute » d'aut
FROM Album LEFT JOIN Groupe ON ...  -- Garde TOUS les albums et y « ajoute » d'autr
```

13. 🎯 Résumé : checklist en bref

- Ordre des clauses respecté (`SELECT` → `FROM` → `JOIN` → `WHERE` → `ORDER BY`).
- Guillemets simples autour des chaînes de caractères.
- Syntaxe `JOIN ... ON` (pas de virgules dans `FROM`).
- Colonnes qualifiées si ambiguïté (`Table.colonne`).
- Chaque `JOIN` a son `ON` correspondant.
- Point-virgule à la fin.
- Mots-clés SQL en MAJUSCULES.
- `ORDER BY` utilisé avec `ASC` (par défaut) ou `DESC` si nécessaire.
- Pour `INSERT` : nommer les colonnes si on ne veut pas spécifier la valeur de la clef primaire.
- Pour `UPDATE` : `WHERE` obligatoire (*après* le `SET`) !